Reverse-Engineering.Info

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□     ____                     __        __              □□□□
□□    /  _/_ _  __ _  ___  ____/ /____ _/ /              □□ □□□
□□   _/ //  ' \/  ' \/ _ \/ __/ __/ _ `/ /               □□ □ □
□□ /___/_/_/_/_/_/_/\___/_/  \__/\_,_/_/                 □□ □ □
□□    ____                          __          __       □□ □ □
□□   / __ \___ ___ ________ ___  ___/ /__ ____  / /____□□ □ □
□□ / /_/ / -_|_-</ __/ -_) _ \/ _  / _ `/ _ \/ __(_-<□□ □ □
□□/_____/\__/___/\__/\__/_//_/\_,_/\_,_/_//_/\__/___/□□ □ □
□□                                                      □□ □ □
□□       Web: http://www.ImmortalDescendants.org        □□ □ □
□□                 Author: [yAtEs]                      □□ □ □
□□                   Date: 12/Sept/00                   □□ □ □
□□           Topic: Creating A Basic VxD                □□ □ □
□□                  (Part I of III)                     □□ □ □
□□               Level: Beginner                        □□ □ □
□□                                                      □□ □ □
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □ □
  □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □
    □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

```
===========================================================================
PUBLISHER'S NOTE: The Kit mentioned in this essay can be found here:
www.reverse-engineering.info/files/vxdkit.zip
===========================================================================
```

Hi I have decided to write 3 tutorials on VxDs, part 1 on creating a basic template, part 2 on INT hooking and part 3 on API hooking. VxDs are quiet simple really *cough* and can be fun to play with, some people don't like the idea of a VxD and like to use them little as possible, but i say...blah ;)

The VMM is a small protected mode program which has the fine job of holding your computer together ;) if you disagree with that just try deleting it..
VMM sends out Control Messages to all loaded VxDs when anything interesting happens so our VxD frame will be based around a Device_Control_Procedure
to handle these events and react in the appropriate way.

Ok before we start the process of creating all things nice and wonderful
you should get the required 'tools' The Windows98 DDK has just about everything
you could want:- http://www.microsoft.com/hwdev/ddk/install98ddk.htm?
but if you live in the UK and can't wait 2 days to download that 20 meg file
here is a small package with the very basic stuff (wow I'm nice;)

Reverse-Engineering.Info

get it at
www.reverse-engineering.info/files/vxdkit.zip (557kb)

Coding a VxD is not like a normal win32 program, all the VxD code must be put into special pre-defined segments these segments are defined in a .DEF file and called using macros in the actual source, the segments are different areas of code/data which we must set-up to tell the compiler if this code/data is going to be pageable or locked for e.g.

A DEF file will have the following format

```
NAME
SEGMENTS
EXPORTS

and now the syntax:-

;___________________________________________________________________________
VXD <NAME> DYNAMIC (<- if its a static VxD remove the word dynamic)
SEGMENTS

   SEGNAME    CLASS  'class_type'   segments_properties
   SEGNAME    CLASS  'class_type'   segments_properties
   etc
EXPORTS

   DDB  ID
;_____________________________________________________________________________
_
```

make much sense?..yer that's what i thought,

ok the VxD name must be in uppercase.

lets look at some class_types

```
LCODE   - Page-locked code and data, this is data and code which will be
          always in memory and never paged to disk, mostly used for
          INT code etc, code that must be present at all time
PCODE    - Pageable code, this is code that can be paged to disk if
           theres a physical memory crisis :)
PDATA    - Pageable data
ICODE    - Initialisation code, this is code that is discarded after the
           init` of the VxD
DBOCODE - Debug-only code/data, contains debug_query control message
SCODE    - static code/data, present in memory even when vxd is unloaded
RCODE    - Real-Mode initialisation code and data, blah blah 16bit for real
mode
16ICODE - USE16 protected-mode initialisation data, contains code that can
be
           copied from protected mode to V86
MCODE    - Locked message strings.
```

vmm.inc has some macros defined for creating different segments in your source code, so the segnames would be as followed:-

SEGNAME	MACRO
_LTEXT	VxD_LOCKED_CODE_SEG
_PTEXT	VxD_PAGEABLE_CODE_SEG
_DBOCODE	VxD_DEBUG_ONLY_CODE_SEG
_ITEXT	VxD_INIT_CODE_SEG
_LDATA	VxD_LOCKED_DATA_SEG
_IDATA	VxD_IDATA_SEG
_PDATA	VxD_PAGEABLE_DATA_SEG
_STEXT	VxD_STATIC_CODE_SEG
_SDATA	VxD_STATIC_DATA_SEG
_DBODATA	VxD_DEBUG_ONLY_DATA_SEG
_16ICODE	VxD_16BIT_INIT_SEG
_RCODE	VxD_REAL_INIT_SEG

so to define a locked segment in our source file we would have an _LTEXT in our def file then in our source file enter:-

```
VxD_LOCKED_CODE_SEG

 CODE HERE

VxD_LOCKED_CODE_SEG_ENDs
```

woo simple eh? anyway the last thing is the exports, you must have an export to the DDB which i will explain later.

here is an example DEF file which can be used in all projects, the unused segments will just create warnings at compile stage.

```
;_____________________________________________________________
VXD FIRST DYNAMIC
SEGMENTS
    _LPTEXT     CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _LTEXT      CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _LDATA      CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _TEXT       CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _DATA       CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    CONST       CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _TLS        CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _BSS        CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _LMGTABLE   CLASS 'MCODE'    PRELOAD NONDISCARDABLE IOPL
    _LMSGDATA   CLASS 'MCODE'    PRELOAD NONDISCARDABLE IOPL
    _IMSGTABLE  CLASS 'MCODE'    PRELOAD DISCARDABLE IOPL
    _IMSGDATA   CLASS 'MCODE'    PRELOAD DISCARDABLE IOPL
    _ITEXT      CLASS 'ICODE'    DISCARDABLE
    _IDATA      CLASS 'ICODE'    DISCARDABLE
    _PTEXT      CLASS 'PCODE'    NONDISCARDABLE
    _PMSGTABLE  CLASS 'MCODE'    NONDISCARDABLE IOPL
    _PMSGDATA   CLASS 'MCODE'    NONDISCARDABLE IOPL
    _PDATA      CLASS 'PDATA'    NONDISCARDABLE SHARED
    _STEXT      CLASS 'SCODE'    RESIDENT
    _SDATA      CLASS 'SCODE'    RESIDENT
    _DBOSTART   CLASS 'DBOCODE'  PRELOAD NONDISCARDABLE CONFORMING
    _DBOCODE    CLASS 'DBOCODE'  PRELOAD NONDISCARDABLE CONFORMING
    _DBODATA    CLASS 'DBOCODE'  PRELOAD NONDISCARDABLE CONFORMING
    _16ICODE    CLASS '16ICODE'  PRELOAD DISCARDABLE
    _RCODE      CLASS 'RCODE'
EXPORTS
    FIRST_DDB  @1
;_____________________________________________________________
; Template from Iczelion`s VxD tutorial set (C) http://win32asm.cjb.net
```

****
The Source File
****

ok lets look at the first bit of a VxD

```
.386p
include vmm.inc
DECLARE_VIRTUAL_DEVICE FIRST,1,0, FIRST_Control, UNDEFINED_DEVICE_ID,
UNDEFINED_INIT_ORDER

Begin_control_dispatch FIRST
End_control_dispatch FIRST
```

```
end
```

;d doesn't look like asm huh, thats cos in reality VxDs are enough to blow your head off in pure asm, so they mostly consist of macros in the inc file.

first line is to set 80386 and privileged instructions, next line is the vmm include with all our macros.

The next line sets up the DDB, the DDB is the Device Descriptor Block and holds information and pointers to various things about the vxd, the DDB has 22 members but we only have to fill a few in, you can see its full structure in the inc file listed as VxD_Desc_Block. the macro Declare_Virtual_Device sets up the DDB in the following format

```
Declare_Virtual_Device Name, MajrVer, MinrVer, CtrlProc, DeviceID, InitOrder,
V86Proc, PMProc, RefData

Name - The name of the VxD in uppercase, this macro appends _DDB to the name and
that
       is the name of the DDB we export in our def file

MajrVer/MinrVer - Major and minor version of your vxd
CtrlProc        - Teh name of your device control procedure, this should be the
vxd name with
                  _Control appended
DeviceID        - unique identifier
InitOrder       - when should your device be loaded? 1st,2nd etc?
V86Proc/PMProc  - address of apis to export for use by V86 and protected mode
programs
RefData         - Referenced data used by IOS
```

after all that we have our message control procedure which will contain our control message handle ;)

ok lets create a source file myvxd.asm

```
;___________________________________________________
.386p
include vmm.inc

DECLARE_VIRTUAL_DEVICE FIRST,1,0, FIRST_Control,\
     UNDEFINED_DEVICE_ID, UNDEFINED_INIT_ORDER

Begin_control_dispatch FIRST
End_control_dispatch FIRST



end
```

;____

also myvxd.def, paste the def file from earlier

now compile with the following cmds

```
ml -c -DMASM6 FIRST.asm
link -vxd FIRST.obj -def:FIRST.def
```

woo and it compiles with lots of warnings about our unused segments,..so
we have a vxd..not much good thou is it, ;), there are a few more things we
need to do and also create a loader, dynamic VxDs are loaded with CreateFileA,
cos i'm a TASM programmer ;d and only use masm for VxDs i created my loader in TASM
so just to complicate things here is my TASM source for a VxD loader.

```
;_____________________________________________________________________
.486P
locals
jumps

.Model Flat ,StdCall

Extrn   MessageBoxA:PROC
Extrn   exitprocess:PROC
Extrn   CreateFileA:PROC
Extrn   CloseHandle:PROC
Extrn   GetModuleHandleA:PROC
Extrn   GetProcAddress:PROC
Extrn   DeviceIoControl:PROC
.data
file1 db "\\.\FIRST.vxd",0
fbox  db   'Loader',0
ftitle db  'you broke it',0
ftitle2 db  'Loaded',0
handle1 dd ?

.code

main:

 Call CreateFileA,offset file1,0,0,0,0,4000000h,0
 cmp eax,-1
 je fuxor
 mov handle1,eax

Call MessageBoxA,0,offset ftitle2,offset fbox,0
jmp endprog

fuxor:
 Call MessageBoxA,0,offset ftitle,offset fbox,0

endprog:
 Call CloseHandle,handle1
 call exitprocess,0

end main
;_____________________________________________________________________
```

ok that should be straight forward enough, once you have it compiled
give it a run with the VxD see what happens.

doesn't work does it, that's because when a VxD is loaded the
w32_deviceIoControl is sent and your VxD must return 0 for
DIOC_Open message.

so now we must learn to cope with the control messages, remember out
control procedure?

Begin_control_dispatch FIRST
End_control_dispatch FIRST

here we must process the messages, to do this we use the macro Control_Dispatch

```
Control_Dispatch MSG,PROC_TO_EXECUTE
```

so now we must add the following code to our control dispatch

```
Begin_control_dispatch FIRST
    Control_Dispatch w32_DeviceIoControl, OnDeviceIoControl
End_control_dispatch FIRST
```

w32_DeviceIoControl is the message sent to the VxD, OnDeviceIoControl is our new
procedure which will handle the message, we must now create a code segment and code
a proc to return 0 to the message, like so:

```
;______________________________________________________________________
VxD_PAGEABLE_CODE_SEG
;______________________________________________________________________

BeginProc OnDeviceIoControl
    assume esi:ptr DIOCParams
    .if [esi].dwIoControlCode==DIOC_Open
        xor eax,eax
    .endif
    ret
EndProc OnDeviceIoControl

;______________________________________________________________________
VxD_PAGEABLE_CODE_ENDS
;______________________________________________________________________
```

Reverse-Engineering.Info

Save your source and recompile, now you will find the VxD loads, click ok and it is unloaded.

```
full source
===========

ASM
===

;_____________________________________________________________________________
.386p
include vmm.inc
include vwin32.inc
include shell.inc

DECLARE_VIRTUAL_DEVICE FIRST,1,0, FIRST_Control,\
     UNDEFINED_DEVICE_ID, UNDEFINED_INIT_ORDER

Begin_control_dispatch FIRST
    Control_Dispatch w32_DeviceIoControl, OnDeviceIoControl
End_control_dispatch FIRST


;________________________________________________________________________
VxD_PAGEABLE_CODE_SEG
;________________________________________________________________________

BeginProc OnDeviceIoControl
    assume esi:ptr DIOCParams
    .if [esi].dwIoControlCode==DIOC_Open
        xor eax,eax
    .endif
    ret
EndProc OnDeviceIoControl

;________________________________________________________________________
VxD_PAGEABLE_CODE_ENDS
;________________________________________________________________________

end
;_____________________________________________________________________________
```

```
DEF
===

;___________________________________________________________________________
VXD FIRST DYNAMIC
SEGMENTS
    _LPTEXT      CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _LTEXT       CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _LDATA       CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _TEXT        CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _DATA        CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    CONST        CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _TLS         CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _BSS         CLASS 'LCODE'    PRELOAD NONDISCARDABLE
    _LMGTABLE    CLASS 'MCODE'    PRELOAD NONDISCARDABLE IOPL
    _LMSGDATA    CLASS 'MCODE'    PRELOAD NONDISCARDABLE IOPL
    _IMSGTABLE   CLASS 'MCODE'    PRELOAD DISCARDABLE IOPL
    _IMSGDATA    CLASS 'MCODE'    PRELOAD DISCARDABLE IOPL
    _ITEXT       CLASS 'ICODE'    DISCARDABLE
    _IDATA       CLASS 'ICODE'    DISCARDABLE
    _PTEXT       CLASS 'PCODE'    NONDISCARDABLE
    _PMSGTABLE   CLASS 'MCODE'    NONDISCARDABLE IOPL
    _PMSGDATA    CLASS 'MCODE'    NONDISCARDABLE IOPL
    _PDATA       CLASS 'PDATA'    NONDISCARDABLE SHARED
    _STEXT       CLASS 'SCODE'    RESIDENT
    _SDATA       CLASS 'SCODE'    RESIDENT
    _DBOSTART    CLASS 'DBOCODE'  PRELOAD NONDISCARDABLE CONFORMING
    _DBOCODE     CLASS 'DBOCODE'  PRELOAD NONDISCARDABLE CONFORMING
    _DBODATA     CLASS 'DBOCODE'  PRELOAD NONDISCARDABLE CONFORMING
    _16ICODE     CLASS '16ICODE'  PRELOAD DISCARDABLE
    _RCODE       CLASS 'RCODE'

EXPORTS
    FIRST_DDB  @1
;___________________________________________________________________________
```

and that is a basic dynamic VxD template, which is what i wanted to show
you, but you probably think that's a bit boring so lets make a blue screen
when to VxD is loaded.

We need to call a new procedure when the VxD is loaded so lets use the
Sys_Dynamic_Device_Init control message and a data segment is also required, I'll just
paste the source it should be self explained, the only you may not know is the parameters
for the VxD services.

```
;_____________________________________________________________________________

.386p
include vmm.inc
include vwin32.inc
include shell.inc

DECLARE_VIRTUAL_DEVICE FIRST,1,0, FIRST_Control,\
     UNDEFINED_DEVICE_ID, UNDEFINED_INIT_ORDER

Begin_control_dispatch FIRST
    Control_Dispatch Sys_Dynamic_Device_Init, BlueScreen
    Control_Dispatch w32_DeviceIoControl, OnDeviceIoControl
End_control_dispatch FIRST

;______________________________________________________________________
VxD_PAGEABLE_DATA_SEG
;______________________________________________________________________

    pmsg db 'Error fault at 31337:0xVXDC0DE',0
    ptitle db 'Warning',0
;______________________________________________________________________
VxD_PAGEABLE_DATA_ENDS
;______________________________________________________________________
;______________________________________________________________________
VxD_PAGEABLE_CODE_SEG
;______________________________________________________________________

BeginProc OnDeviceIoControl
    assume esi:ptr DIOCParams
    .if [esi].dwIoControlCode==DIOC_Open
        xor eax,eax
    .endif
    ret
EndProc OnDeviceIoControl

BeginProc BlueScreen

 mov edi,offset ptitle
 mov ecx,offset pmsg
 mov eax,MB_OK
 VMMCall Get_Sys_VM_Handle
 VxDCall SHELL_sysmodal_Message
 clc
 ret
EndProc BlueScreen
;______________________________________________________________________
VxD_PAGEABLE_CODE_ENDS
;______________________________________________________________________

end
```

Republished - 12th November 2007 – Robert Yates

```
;_______________________________________________________________________________________
```

compile and run, OOoh amazing huh?, that's it then, till next time when we look at Interrupt hooking.

special greetz to Iczelion,Defiler,{sMaEgLe},_risc,Noodlespa

any problems or mistakes feel free to contact me and i shall assist ;)

[yAtEs]
"Keep it locked, keep it hardcore. Roots 'n' phuture. Peace."